Aufgabe 1

Ändern Sie die Sprache **WHILE** ab, indem Sie anstelle des atomaren Ausdruckes `read` Anweisungen der Form `read I` zulassen. Die Semantik dieser Anweisung lautet informell: Die Ausführung von `read I` bewirkt eine Zuweisung des nächsten Eingabewertes an die Variable `I` und eine Verkürzung der Eingabedatei um das erste Element.
Formalisieren Sie die Semantik von `read I` denotationell.

Idee von Flo:

C[[read I]] (s,e,a) = (S[n/I],e',a) mit e = n.e'

Soll heißen: der nächste gelesene Eingabewert (n) ersetzt (I) im Speicher. (e') ist ein um den ersten Wert kürzeres (e). Naja....

Aufgabe 2

Erweitern Sie die Sprache **WHILE** um Anweisungen der Form

```
for I := T_1 to T_2 do C
```

. Formalisieren Sie die Semantik dieser Anweisungen denotationell.

Idee von Flo:

Das soll doch am Ende ne ganz simple Zählvariable werden denk ich mal, also:

C[[while I:/=T2 do C]] (s,e,a) = C[[C; while I:/=T2 do C; I:= I+1]](s,e',a)

Solange I noch nicht T2 entspricht, wird das Programm weiter ausgeführt.

Aufgabe 3

Erweitern Sie die Sprache WHILE um den atomaren booleschen Term `eof`. Die informelle Semantik von eof lautet: eof ist wahr gdw die Eingabe leer ist.
Formalisieren Sie die Semantik von `eof` denotationell.

Idee von Flo:

B[[eof]] s,e = (falsch, e'), falls e = b.e' mit b e BOOL oder e = E oder e=n.e' mit n e Zahl
B[[eof]] s,e = (wahr, e'), sonst

Ich hoffe ich habe alle möglichen Eingabearten abgegriffen.

Aufgabe 4

Programmieren Sie in WHILE (einschließlich `eof`) einen Algorithmus zur Berechnung der Summe aller Eingabewerte. Beweisen Sie die Korrektheit Ihres Programms anhand der denotationellen Semantik. Diskutieren Sie die Problematik beim Fehlen von `eof`.